Introduction to Operating Systems

Prof. Pai H. Chou
National Tsing Hua University

Agenda

- Prerequisites
- A Historical Perspective
- Administrative
- Course Materials
- Approach
- Schedule

Prerequisites

Prerequisites

- Hardware
  - Computer architecture, input/output, interrupts
- Software
  - assembly programming: registers, stacks
  - system programming language (C)
  - fundamental data structures (arrays, stacks, linked lists, trees)

Prerequisite: C language

- Scoping
  - global vs. (file) static, auto local vs. static local
- Pointers
  - Pointer type, pointer expression vs. variable
  - Pointer to array, structs, function, pointer arithmetic
- Memory allocation
  - stack allocation vs.heap allocation (malloc(), free())
- Bit manipulation

Prerequisites: How C works under the hood

- Compiler-Assembler-Linker-Loader
  - User code, library code, system call
- Stack
  - Stack frame
  - Parameter passing and return value
  - Function return address
  - Saving and restoring register values across calls

Prerequisites: Unix Tools

- Unix shell (bash, csh) basic commands
- Compiler, linker, debugger
- Editor
- Makefile
- Version control
- Scripting language

Prerequisites: Data Structures

- Arrays
- Linked lists
  - singly vs doubly linked, insert, delete, find
- Stacks and Queues
  - implementation using array vs. linked list
- Trees
  - n-ary tree, tree traversal, recursion
- Hash tables
  - hashing, lookup, collision, deletion

Prerequisites: Computer Architecture

- Instruction set architecture (ISA)
  - opcode, operand, register, addressing modes
  - assembly language, machine code,
- Memory organization
  - memory hierarchy: cache, memory, disk
  - stack, heap, architecture support for languages
- Control transfer
  - interrupts, trap, input/output

Overview

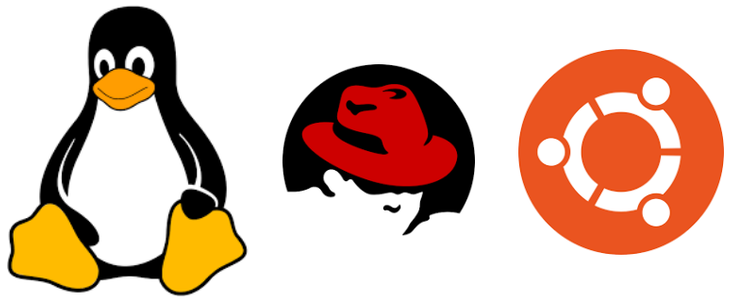
- What is an OS?
- Different kinds of OS
- Problems solved by an OS
- Components of an OS
- Metrics for evaluating an OS
- Trends in OS: multiprocessor, multicore,

You may have heard of

- Microsoft Windows
  - Windows XP, Vista, 7, 8, 10, …

- Apple
  - macOS, iOS, watchOS, tvOS, …

- Gnu/Linux
  - RedHat, Ubuntu, Debian, SUSE, …

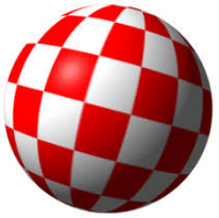
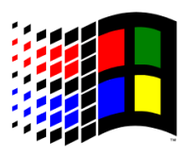


- Google Android
  - KitKat, Lollipop, Marshmallow, Nougat, Oreo, …

You may not have heard of...

- Workstations
  - SunOS, Solaris, DEC VMS, Ultrix, AIX
- Early personal computers, text mode
  - CP/M, Apple DOS, Atari DOS, PC DOS...
- PC in 1980-90s (GUI)
  - IBM OS/2, BeOS, AmigaOS
  - Windows 3.1, 95, 98, ME, NT
- Mobile
  - PalmOS, NewtonOS (PDA), Windows CE
- and may more...



What is an OS?

- Runtime-support software
  - above hardware
  - below application software
- Manages all aspects of computer
  - Execution of programs (scheduling, concurrency)
  - Memory usage (allocation, sharing, mapping)
  - Data Storage, input/output (network, display, keyboard, mouse, touch, audio, video, timing)
  - Protection and Security

OS: the most powerful software

- Whoever controls the OS rules the world!
  - Microsoft Windows (plus Office) made Bill Gates the richest person on earth (until recently)
  - iOS (+iPhone) made Apple the biggest company
  - Android gives Google the widest smartphone user base
- Why? because OS defines the “computer”
  - programmer usually can’t access underlying hardware w/out going through OS
  - OS provides essential service (e.g., GUI, location, …)

But there are many types of "Computer Systems"

- General Purpose
  - PC, tablet, smartphone, server,
- Embedded systems
  - dedicated purpose; normally don't think of as computer
    e.g., iPod, DVD player, washing machine, elevator controller, ..
  - has processor, runs software / firmware
  - may have storage, communication, …
- More units of embedded systems in use
  - example: a car may have > 100 processors!
  - more coming in the Internet of Things (IoT)

Example: Tesla Model S or X

65 processors!

- 1 Anti-lock Braking System
- 2 AC Motor Inverter / Charger
- 1 Active Cruise Control
- 1 Air Suspension
- 1 AM/FM/HD radio
- 1 Audio
- 1 Auto Pilot 1 Camera
- 1 Auto Pilot 1 Processor
- 2 Auto Pilot 2 Processors
- 16 Battery Sub-Modules
- 1 Bluetooth
- 2 Charger
- 1 Connectivity
- 4 Door Handle Control
- 1 Folding Mirror
- 1 FOB receiver
- 1 GPS
- 1 HVAC
- 1 Homelink
- 1 Instrument Display
- 1 Main Display
- 1 Mobile charger
- 1 Parking brake
- 1 Parking sensor
- 1 Power Steering
- 1 Rear Camera
- 2 Rear Taillight
- 1 Safety Restraints
- 2 Seat Controller1 Security module
- 4 Side Window Controller
- 1 Sunroof controller
- 1 Thermal controller
- 1 Tire Pressure Monitor
- 1 Traction Control
- 2 Body Control Unit
- 1 Wiper control
- 1 XM Satellite

https://teslatap.com/undocumented/model-s-processors-count/

What OS do different processors run?

- Mobile OS
  - carOS (Apple), Android Auto, QNX, Windows Embedded Automotive
- Real-time OS
  - e.g., vxWorks, FreeRTOS, QNX, mBED, ...
- vendor-specific runtime support for protocol stack
  - TI OSAL, Nordic SoftDevices, ...
- No OS ("bare metal")
  - just application code and driver! maybe some library routines for memory, timing, ..

Do you really need an OS?

- Not essential for some embedded systems
  - CPU just runs code; doesn't really "know" it's running an OS
- Language or library support
  - Useful to support concurrency, timing, memory, storage, communication
- Reasons to leave out the OS, esp. some embedded systems
  - Overhead: memory space, slower speed
  - Language + library already provide support
  - Determinism: ability to reproduce the behavior each time
- But... without OS, code gets messy, hard to maintain

Key Issues for this course

- Concurrency
  - Race conditions, Deadlocks, …
- Resource Management
  - Arbitration vs. Virtualization
- System Design
  - Abstraction vs. Monolith
  - Policy vs. Mechanism
- Performance Analysis

Appreciation for Good OS

- What makes an OS good?
  - Bug-free: OS should not have bugs!
  - Robust: OS should not crash
  - Low overhead: ideally takes no memory or time
  - Fair: all processes can run w/out waiting too long
  - Secure: protects the user and system from attacks
  - Helpful: OS provides the right kind of service to user and programmer
- OS often gets the first blame when things don't work right!

A Historical Perspective

Historical Perspective

Mainframes

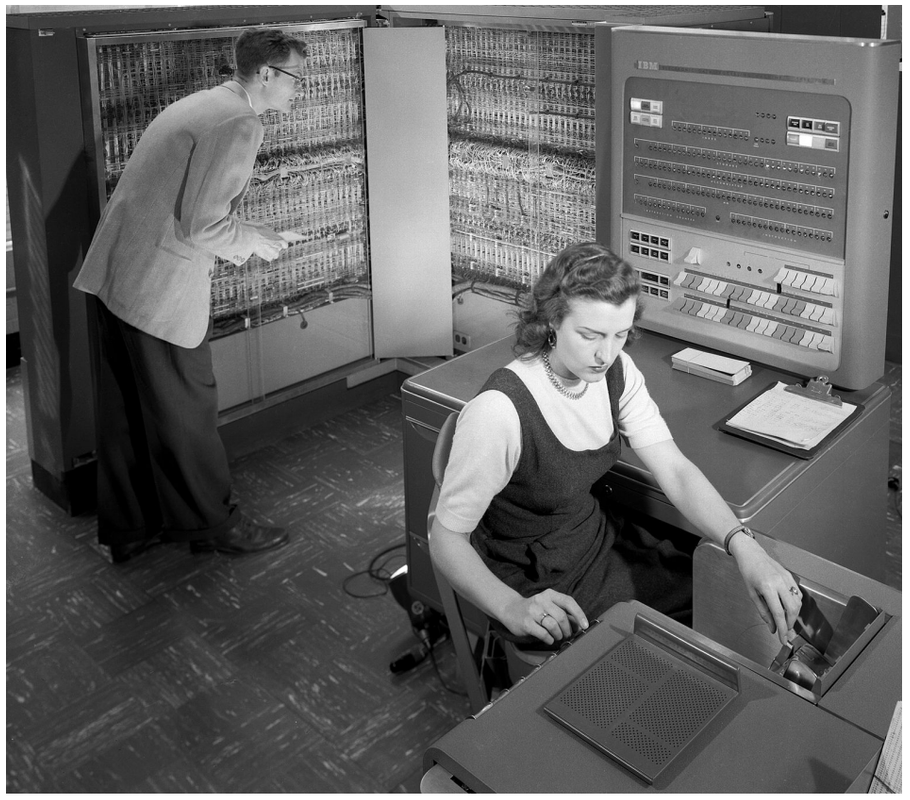
- Batch system
- Multiprogramming
- Time-sharing

Computer-System Architectures

Special-Purpose Systems

1. Mainframe Systems

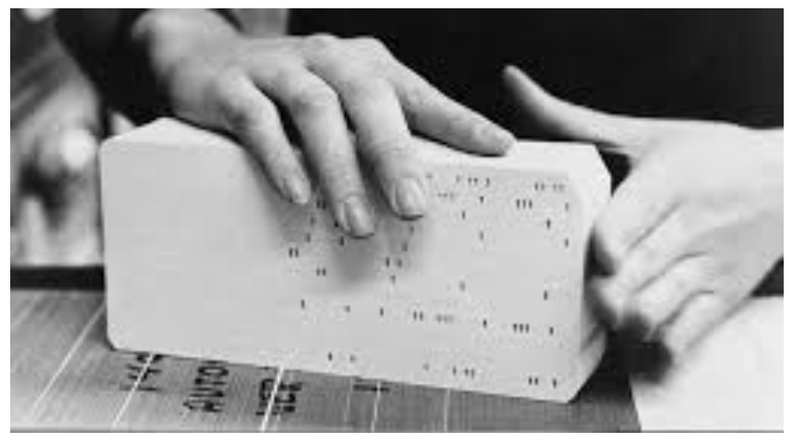
- One of the earliest computers
  - Slow I/O devices: card reader, printer, tape drives
- Evolution:
  - Batch -> Multiprogramming -> Time-shared
- Still in use today, but very different



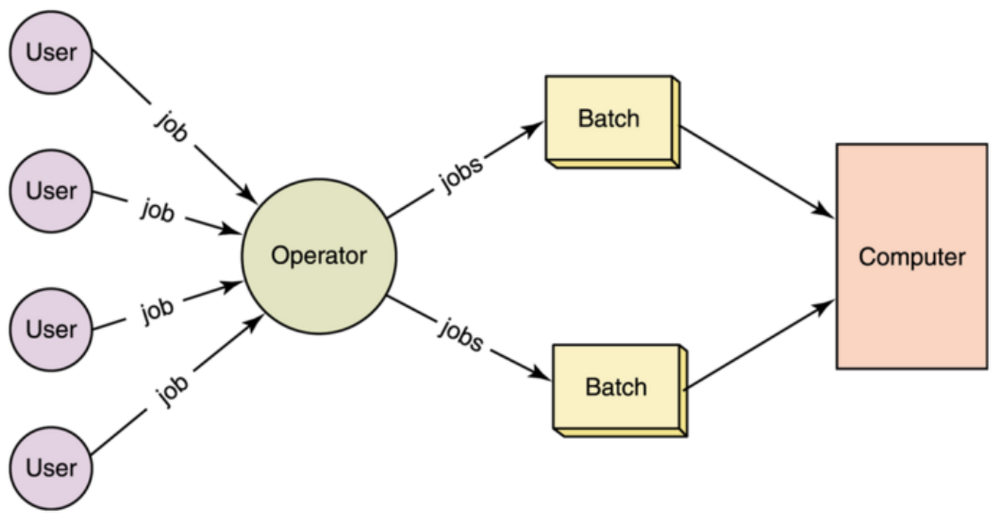
IBM 704 in 1957

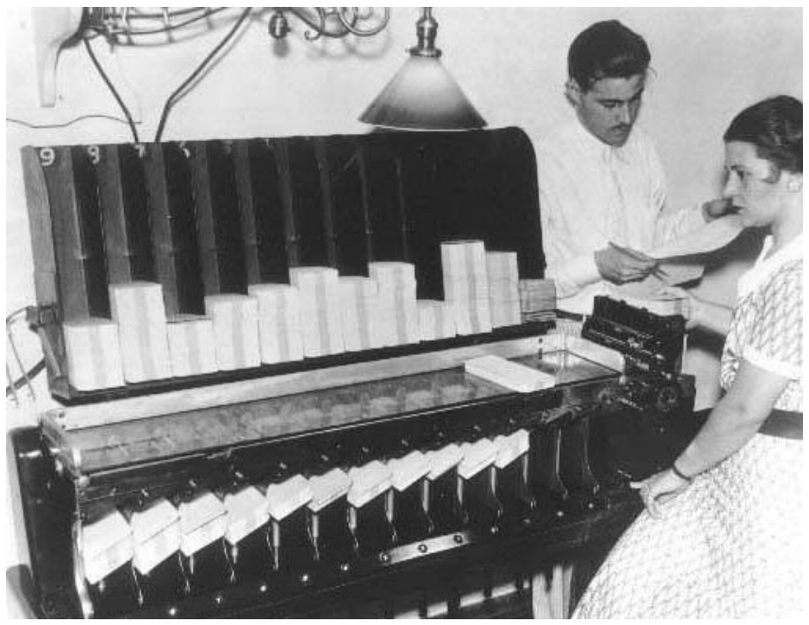
1.1 Mainframe - Batch Systems

- user submit *jobs*
  - program + data on punch cards
- human operator sorts jobs



- OS loads and runs one job at a time





1.3 Time-sharing (multitasking) System: OS Tasks

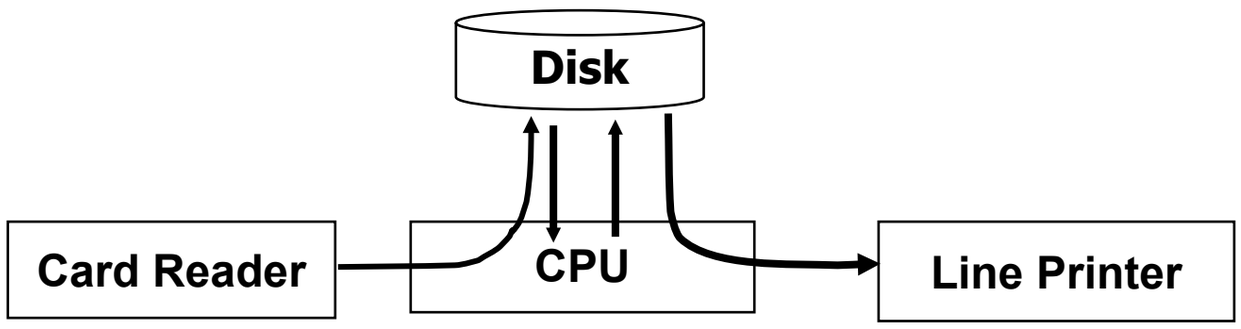
- Virtual memory (ch. 10)
  - swap memory in/out from/to disks to support job demand
- Storage (ch.11), file system (ch.13-14)
  - manages data storage and organization on disk
- Synchronization (ch. 6-7), deadlock (ch. 8)
  - support concurrent execution of program
  - analyze, prevent, detect, and resolve deadlocks

1.1 Mainframe - Batch Systems

- Advantages
  - Repeated jobs are done fast without user interaction.
  - Offline makes less stress on processor
  - Sharing system for multiple users
  - You can assign specific time for the batch jobs
- Drawbacks
  - one job at a time
  - no interaction between users and jobs
  - CPU is often idle: I/O speed much slower than CPU speed

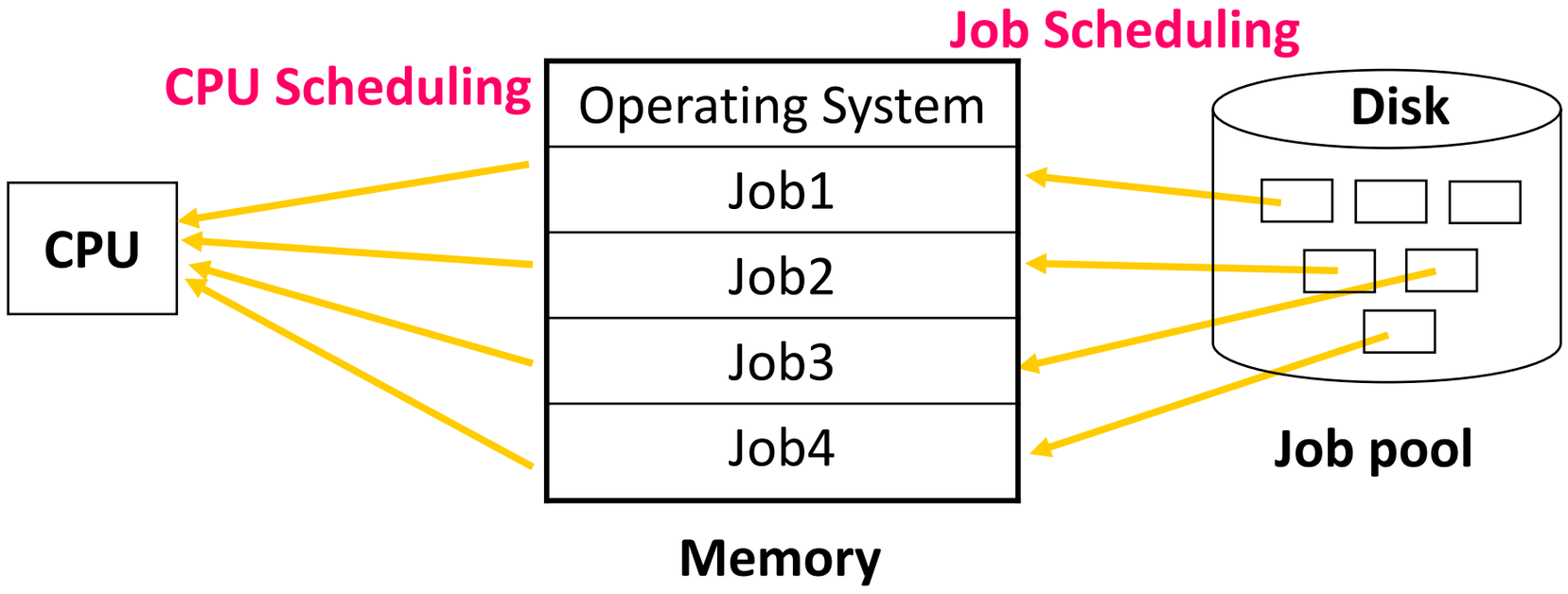
1.2 Mainframe - Multipgramming

- Overlap I/O computation of jobs
  - keeps both CPU & I/O devices utilized in parallel
- Spooling
  - “simultaneous peripheral operation online”
  - I/O spooled to disk, free up CPU to do processing
  - CPU just needs to be notified when I/O is done



1.2 Mainframe - Multiprogramming

- Several jobs are loaded into memory
- CPU is *multiplexed* among jobs
- I/O may be *spooled* or *interactive*



1.2 Mainframe - Multiprogramming OS tasks

- Memory Management (ch. 10)
  - system allocates memory to multiple jobs
- CPU scheduling (ch. 5)
  - system chooses which job to run and for how long
- I/O system (ch. 12)
  - runtime support in terms of device drivers
  - allocation of devices to jobs

1.3 Mainframe - Time-sharing (multitasking) System

- An *interactive* system between user & system
  - CPU switches among jobs quickly to support interaction
  - User can see result immediately (response time < 1s)
  - Usually, keyboard + text display
- Context switch among multiple users
  - job completion
  - waiting on I/O
  - after some short execution time

1. Mainframes: Summary

	1.1 Batch	1.2 Multiprogramming	1.3 Time-shared
System model	Single user, single job	multiprogramming	Multiple user, multiple programs
Objective	Simplicity	Resource utilization	Interactivity, Responsiveness
OS features	n/a	CPU scheduling, memory mgmt, I/O system	File system, virtual memory, synchronization

Historical Perspective

Mainframes

Computer-System Architectures

- Single processor
- Tightly-coupled multiprocessor
- Loosely-coupled distributed system

Special-Purpose Systems

2.1 Single-Processor

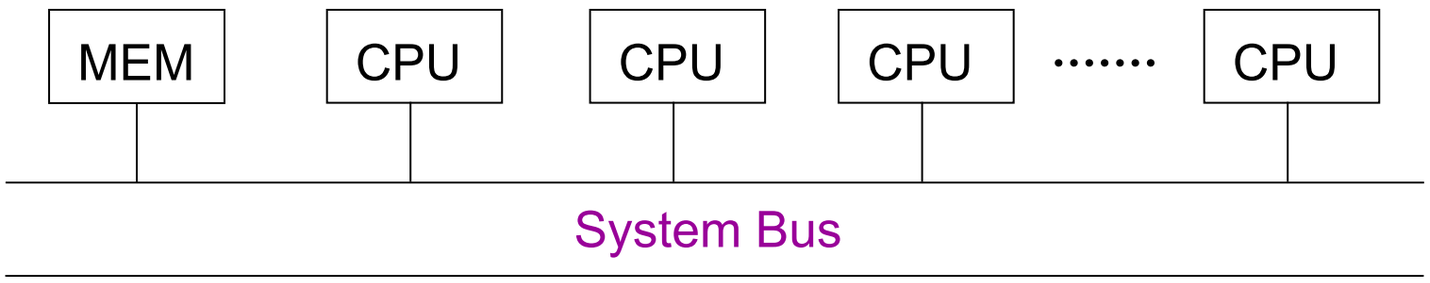
- Traditional personal computers
  - Usually emphasizes interactivity
  - I/O devices: keyboard, mouse, monitor, printer, …
- Previously OS offered no protection
  - DOS, Windows 3.1, Windows 95/98/ME, MacOS (1-9)
  - program could crash whole system, prone to virus
- Modern PC OS's are multitasking w/ protection
  - OS2, Windows NT/XP/…, macOS, Linux / Android

2.1 (single-processor) ISAs

- SPARC (Sun Microsystems) (Berkeley RISC)
- MIPS (MIPS) (Stanford MIPS)
- Alpha (Digital Equipment Corporation)
- PowerPC (Apple, IBM, Motorola)
- 8086, 80286, 80386, 80486, (x86), Pentium, … (Intel)
- Motorola 68000, 68030, 68040, ...
- ARM, Thumb, Thumb-II, ...
- RISC-V (Berkeley)

2.2 Parallel Systems

- Also known as
  - *multiprocessor* or *tightly coupled* systems
- Multiple *cores* in close communication
  - usually communicate through shared memory
- Purpose
  - throughput, economical, reliability, power efficiency

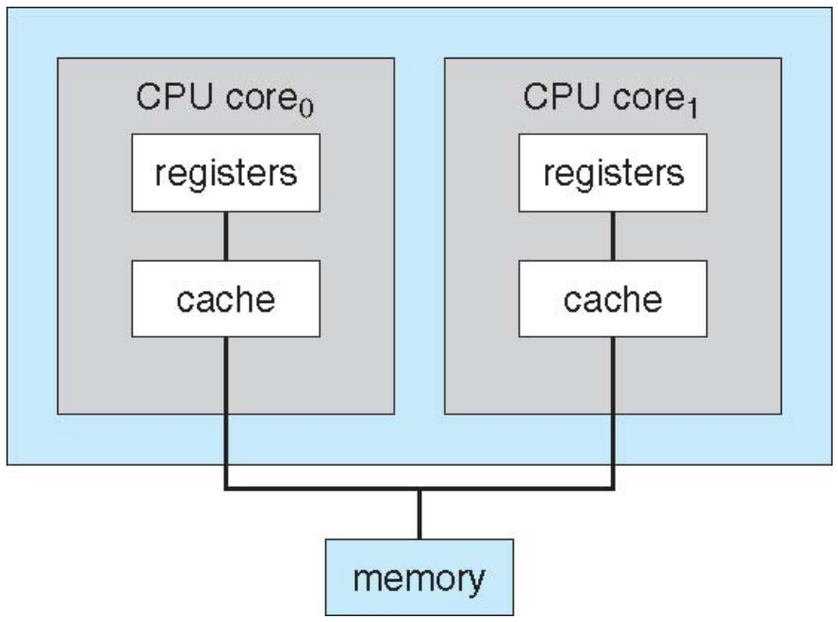


2.2 Parallel Systems

- *Symmetric* multiprocessor (SMP)
  - all processors run the same OS
  - most popular today (e.g, Intel i3/i5/i7)
  - requires extensive synchronization
- *Asymmetric* multiprocessor system
  - each processor is assigned a specific task
  - one master CPU and multiple slave CPUs
  - more common in very large systems

2.2 Multi-Core Processors

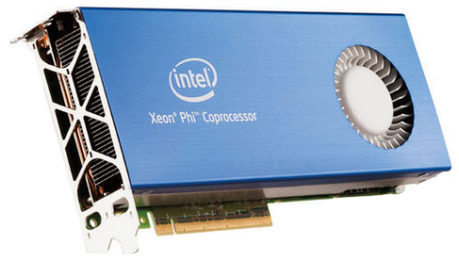
- multiple CPU cores on the same chip
  - on-chip communication is faster than off-chip
- Energy efficiency
  - running a multicore at slower clock (and voltage) => same performance but much lower power!



2.2 Many-Core Processor

- GPGPU - general-purpose graphics processing units
  - e.g., nVidia
  - 2880 thread processor, 1.43 TFlops

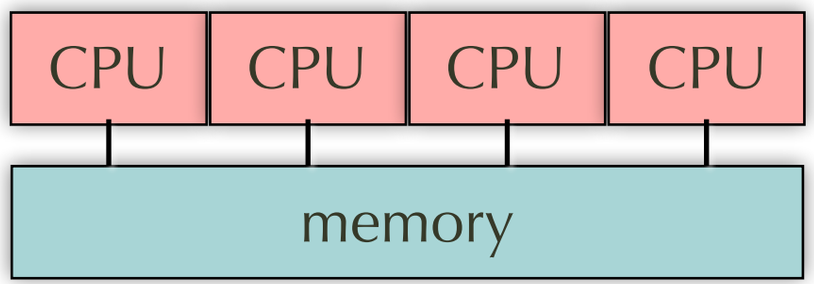
- Intel Xeon Phi
  - Intel Many Integrated Core (MIC)
  - 61 cores, 1.2 TFlops



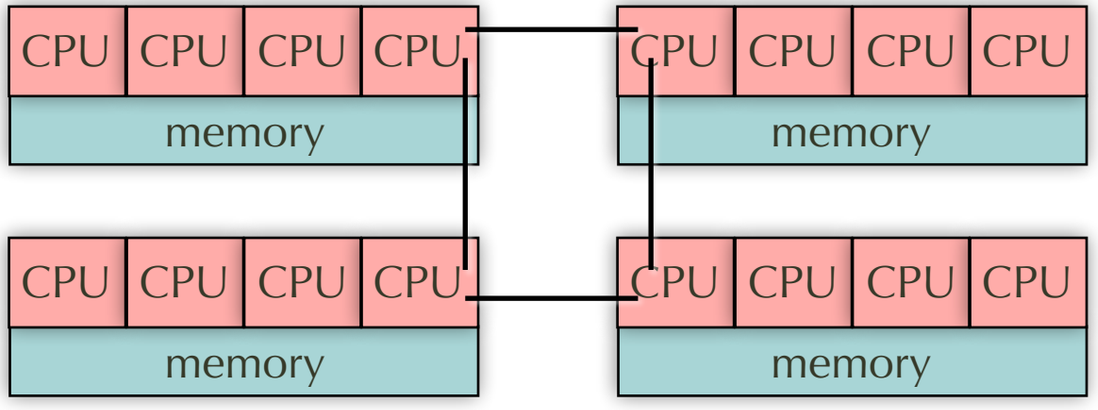
- TILE 64
  - mesh network of 64 tiles; each tile = a general purpose processor

2.2 Memory Architectures

- Uniform Memory Architecture (UMA)
  - most commonly represented by SMP machine today (identical processors)
  - equal access times to memory



- Nonuniform Memory Architecture (NUMA)
  - often made by physically linking multiple SMPs
  - one SMP can directly access memory of another SMP
  - example: IBM blade server

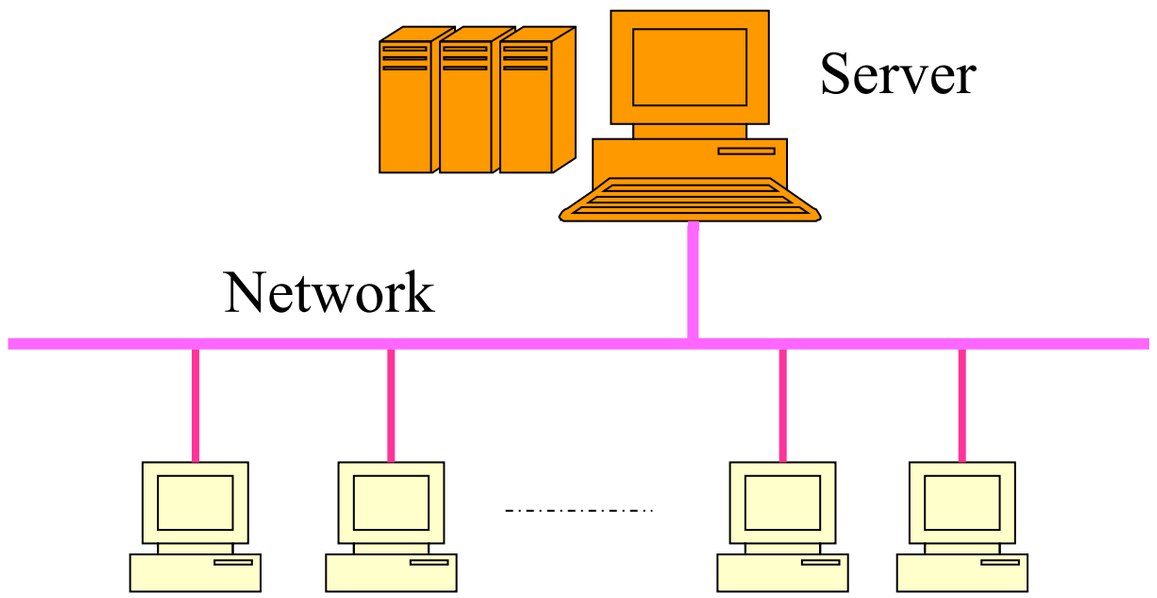


2.3 Distributed Systems

- also known as *loosely coupled* system
  - each processor has its own *local memory*
  - processors communicate with each other through bus or network
  - easy to scale to large number of nodes
- Purposes
  - Resource sharing, Load sharing, Reliability
- Architecture
  - peer-to-peer or client-server

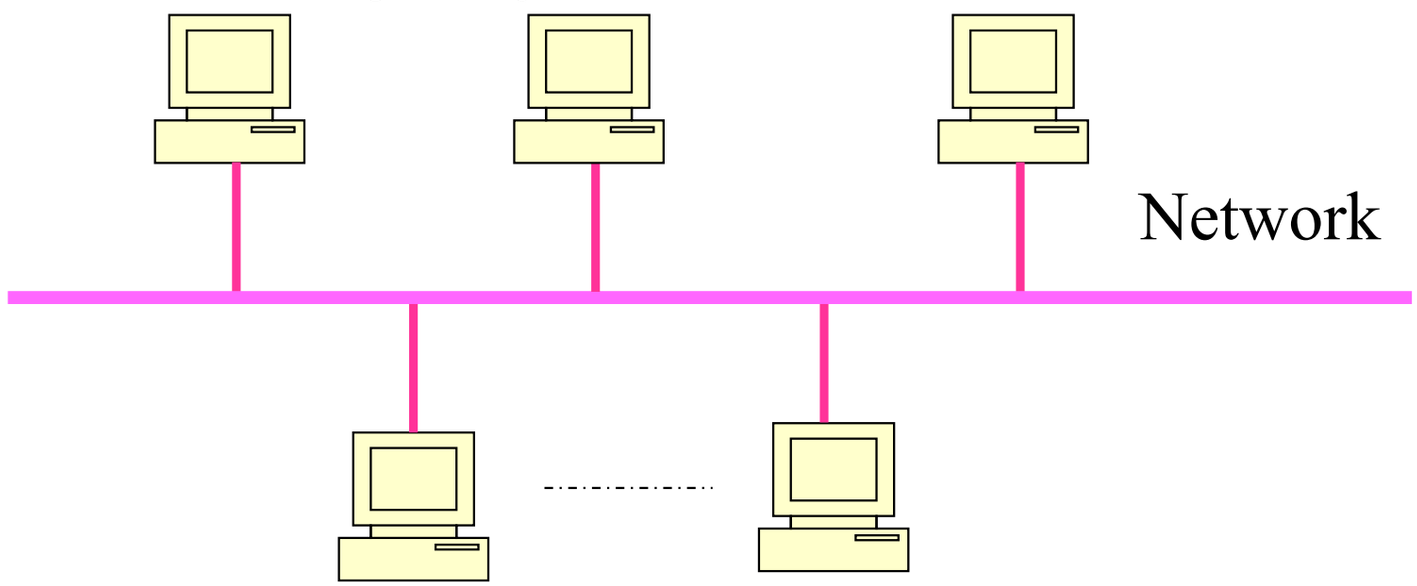
2.3.1 *Client-Server* Distributed System

- Easier to manage and control resources
- However
  - server becomes bottleneck
  - server becomes single point of failure



2.3.2 *Peer-to-Peer* Distributed System

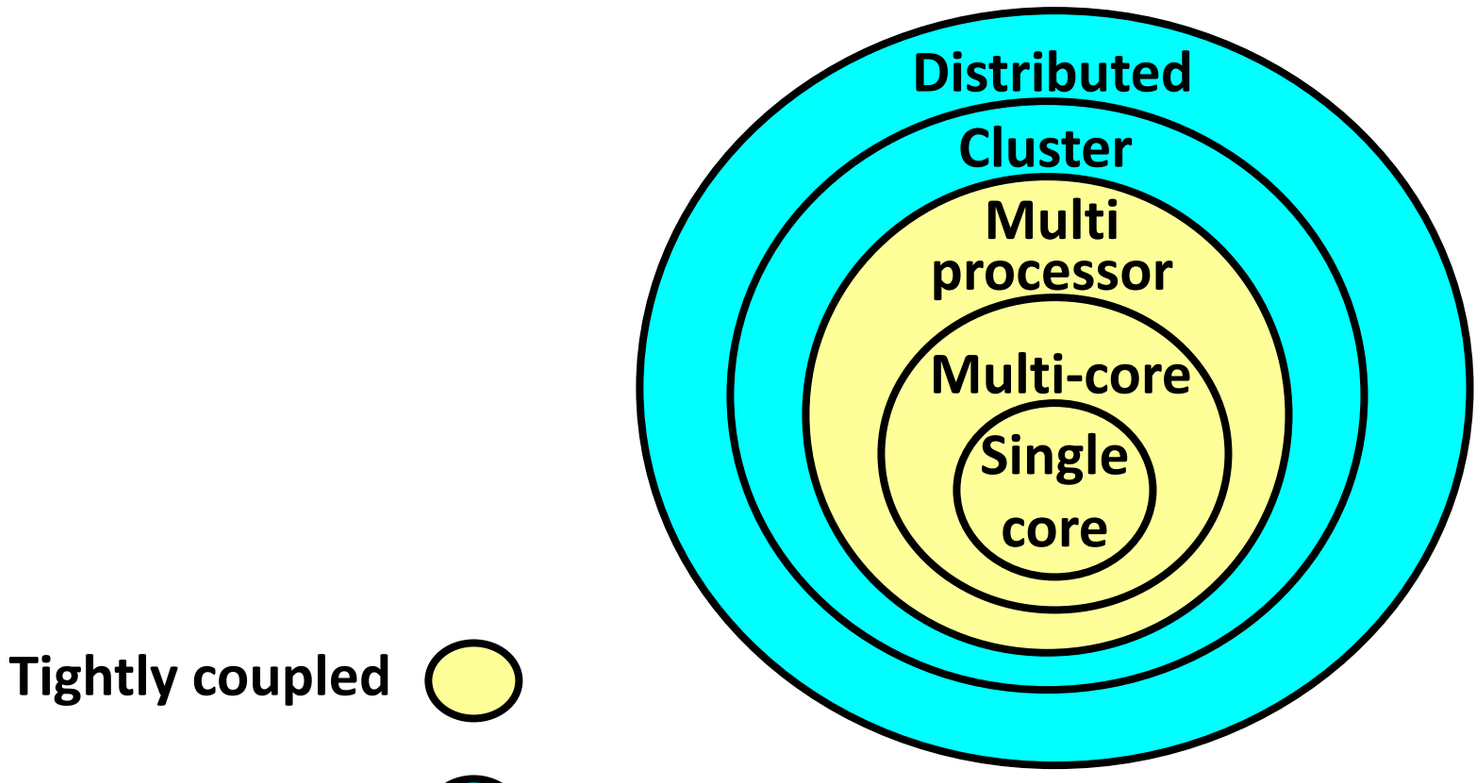
- Every machine is identical in its role in the distributed system
- Example
  - ppStream, bitTorrent, the Internet
- Any system can play the role of client or server



2.3.3 Clustered Systems

- Cluster
  - shared storage
  - closely linked by high-speed LAN (InfiniBand - 300 Gb/s, Myrinet)
    - Examples: Beowulf, http://www.beowulf.org/overview/faq.html
- Asymmetric clusters
  - one server runs application, other servers standby
- Symmetric clusters
  - two or more hosts run applications and monitor each other

2. System Architecture Summary



Tightly coupled

Loosely coupled

Historical Perspective

Mainframes

Computer-System Architectures

Special-Purpose Systems

Real-time systems

Multimedia systems (“soft real-time”)

Handheld systems

3.1 Real-Time Systems

- ability to meet timing constraints
  - absolute or relative deadline, periodic or aperiodic
  - minimum/max time separation
- Does not necessarily mean “fast”
  - guarantee timing constraints are met
- Many applications
  - multimedia system, industrial control, flight or auto control, anti-lock brake

3.1 Soft vs Hard Real-Time

- Soft real-time requirements
  - Undesirable to miss deadline, but not critical
  - example: multimedia streaming
  - Solution: give priority to critical real-time task until it completes
- Hard real-time requirements
  - Missing deadline results in fundamental failure
  - example: nuclear power plant controller => meltdown
  - Solution: remedial action to minimize damage
- OS is NOT required, but makes it more structured and easier to build a real-time system

3.2 Multimedia Systems

- Mostly audio and video
  - e.g., ppstream, online TV, digital broadcast
- Issues
  - timing constraint: 24-30 frames per second
  - on-demand & live streaming
  - compression and decompression before/after streaming
- mostly Soft Real-Time
  - Conventional OS could work if hardware is fast enough or can support certain operations

3.3 Mobile and Embedded Systems

- General-Purpose
  - personal digital assistants (PDA), smartwatches
- Special-Purpose
  - Health-fitness band, smart home switches,
- Considerations
  - Power consumption
  - Limited memory, limited processor performance

Role of OS in different computer systems?

- Runtime Support
  - Do you even need an OS? Or is language/library support sufficient?
- Need for stable structure?
  - Ability to update software?
- Which OS to use?
  - features considerations
  - compatibility considerations

Review

- Mainframe systems
  - Batch, multiprogramming, time-sharing
- Tightly coupled vs Loosely coupled
- UMA vs NUMA
- Distributed systems
  - Client-server vs. P2P
- Real-time
  - Soft vs. Hard Real-time

Administrative

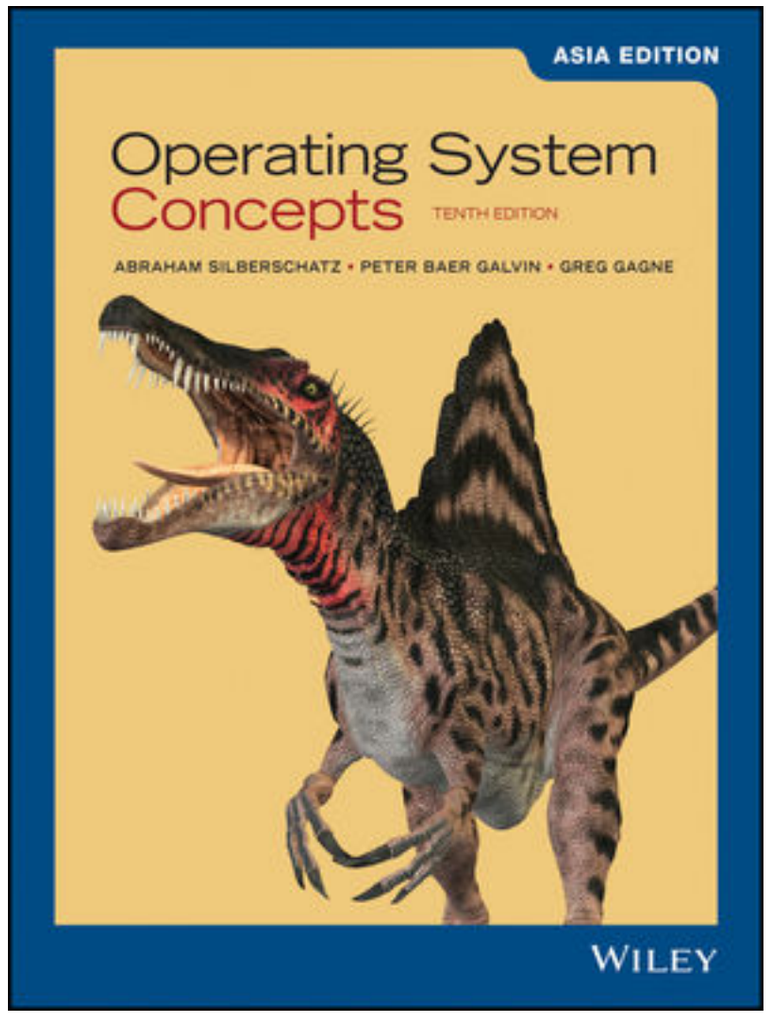
- Instructor
  - Prof. Pai H. Chou, phchou@cs.nthu.edu.tw
  - Office: 620 Delta Building
- TA
  - 宋瑞豐, 方晟軒, 羅仕翰, 吳彥璋, 陳柏宇
- Course website - important!
  - http://lms.nthu.edu.tw/course.php?courseID=40576

Approach

- Topics
  - Following order in the textbook
  - Supplemental materials
- Homework assignments
  - Word problems - write on your own
  - Implement OS algorithms in Python
  - Project: simple OS for an actual processor

Course Materials

- Textbook:
  - Silberschatz, Galvin, and Gagne, *Operating System Concepts, Tenth Edition*, Asia Edition, Wiley, 2019. ISBN 978-1-119-58616-6



- Course website
  - http://lms.nthu.edu.tw/course.php?courseID=40576
  - Lecture slides
  - assignments & course projects

Schedule by the Week

1. Welcome
2. (Ch.1) Introduction
3. (Ch.2) OS structure
4. (Ch.3) Processes
5. (Ch.4) Threads
6. Threads (cont'd)
7. (Ch.5) CPU scheduling
8. (Ch.6-7) Synchronization
9. (Ch.8) Deadlocks
10. (Ch.9) Main Memory
11. (Ch. 10) Virtual Memory
12. (Ch. 11) Mass Storage
13. (Ch. 12) I/O Systems
14. (Ch. 13) File System
15. (Ch. 14) Protection, (Ch. 15) Security

Software

- Python 3
- EdSim51 (requires Java)
- SDCC (Small Device C Compiler)
- GIT (version control)
- Google Suite for Education

Grading Policy

- Breakdown
  - 35% project (OS, due end of semester, demo)
  - 25% midterm
  - 35% final
  - 5% discretionary (participation, quiz, etc)
- Problem set
  - Not graded. Do it on your own
  - Expect questions to show on exam

Programming Project

- To write parts of an operating system
  - write in low-level C and assembly language
  - compile and run code in an ISA simulator
  - use of hardware features (interrupt, timer etc)
- Intermediate milestones every couple of weeks
  - tutorial in nature, self-checked but not graded
  - start from bare metal, gradually add runtime support
- Due by the end of the semester
  - submit complete code and demo

To Do

- Set up G Suite for Education - for submitting homework
  - https://net.nthu.edu.tw/2009/gapp
- Set up own Unix-like environment (if not already)
  - Linux, macOS, CygWin, bash on Ubuntu on Windows 10
- Install basic development tools if not already
  - (g)make, git client, python 3, Java
  - SDCC: (sdcc.sourceforge.net), Edsim51 (edsim51.com)
- Alternatively, download vmware image (1.9 GB)
  - https://drive.google.com/file/d/1aYSz33AAHzHlFSzysGe00NM5jo_Td4YL/view

Tips for this course

- Attend lecture, read textbook before the lecture
  - Don’t just read the slides!
- Internalize the concepts
  - don't just memorize - know why do each step
- Test out your idea
  - high-level: Python
  - low-level: C/assembly, simulator, actual board...